Theoretical Computer Science 47 (1986) 315-321
North-Holland

NOTE

ON THE COMPLEXITY OF PARALLEL PARSING OF GENERAL CONTEXT-FREE LANGUAGES

Wojciech RYTTER*

Department of Computer Science, University of Warwick, Coventry CV4 7AL, United Kingdom

Communicated by D. Perrin
Received April 1986
Revised September 1986

Abstract. Let $T(n)$ be the time to recognize context-free languages on a parallel random-access machine without write conflicts (P-RAM) using a polynomial number of processors. We assume that $T(n)=\Omega(\log n)$. Let $P(n)$ be the time to compute a representation of a parsing tree for strings of length n using a polynomial number of processors. Then we prove $P(n)=O(T(n))$.

A related result is a parallel time $\log n$ computation of the transitive closure of directed graphs having special structure.

The problem of parsing for context-free languages (cfl's, for short) seems to be harder than the problem of recognition. It was proved by Ruzzo [1] that if $T'(n)$ is the time to recognize cfl's on a RAM (sequentially), then the time needed to parse cfl's on a RAM is $O(T'(n)\log n))$. We shall show that when one considers parallel time, then parsing is not harder than recognition (however, the number of processors can grow considerably, though polynomially).

Our model of parallel computation is a parallel random-access machine without write conflicts (known also as a CREW P-RAM). Such a machine consists of a number of synchronously working processors (RAM's) which are using a common memory. No two processors can attempt to write in the same step into the location; however, many processors can read from the same location. Such a model corresponds to bounded fan-in circuits.

The best algorithms for parallel general context-free recognition on a P-RAM work in $\log^2 n$ time using $O(n^6)$ processors; see [2, 3] (such a complexity can even be achieved on much weaker models of parallel computations, cube connected computers and perfect shuffle computers, see [4]). It is a hard open problem whether general context-free recognition can be done in parallel time $\log n$ on a P-RAM. For unambiguous languages, $\log n$ time is enough (see [5]) and if the language is a bracket cfl, then the number of processors can be linear (see [6]). Optimal ($\log n$

* On leave from the Institute of Informatics, Warsaw University, Warsaw, Poland.

time and $n/\log n$ processors) parallel parsing and recognition algorithms can be constructed for one-sided Dyck languages. The algorithms for general context-free recognition use the parsing matrix (to be defined later). We shall show that when one uses this matrix, a parsing tree can be constructed in $\log n$ time using a cubic number of processors. Even if the recognition does not construct the parsing matrix, then we can construct it by executing simultaneously $O(n^2)$ parallel recognizing algorithms. The parallel time does not increase, the number of processors however should be multiplied by n^2 in this case.

Throughout the paper we shall assume (for ease of exposition) that the grammars are in Chomsky normal form. Let G be a context-free grammar in Chomsky normal form, let N and Ter denote the set of nonterminal and terminal symbols, respectively. We write $A \to B$ if the grammar has such a production, and $A \to^* w$ iff the string w can be derived from A. Let S be the starting symbol of the grammar.

Let $w = a[1]a[2]...a[n]$ be a given input string of length n. The recognition problem is to decide whether $A \to^* w$, where $A \in N$. The parsing problem is to construct a parsing tree PT. This tree has $2n-1$ nodes numbered $1, \ldots, 2n-1$. With each node x there is associated the following information:

Father$[x]$, Left$[x]$, Right$[x]$ (left and right sons if x is not a leaf)

Label$[x]$ (an element of N).

The tree should locally satisfy the rules of the grammar:

Label[root] $= S$;

Label$[x] \to$ Label[Left$[x]$] Label[Right$[x]$] if x is not a leaf;

Label$[x] \to a(i)$ if x is the ith leaf (from the left);

Father[Left$[x]$] = Father[Right$[x]$] $= x$.

It is enough to compute only the tables Father and Label. Left and Right can then be easily computed on a P-RAM in $\log n$ time using a small number of processors. On the other hand, if we have Left and Right, then this does not determine Father since PT might be any directed acyclic graph whose non-leaf nodes have outdegree 2.

The parsing table Tab is of the type array $[0, \ldots, n, 0, \ldots, n]$ of subsets of N, Tab$[i,j]$ = **if** $i<j$ **then** $\{A : A \to^* a[i+1]...a[j]\}$ **else** $\emptyset$.

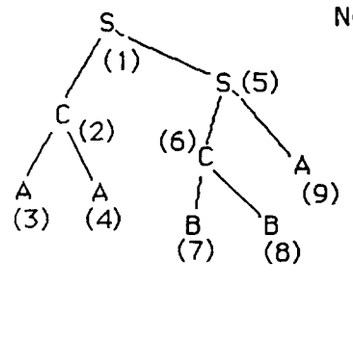
Example. Let G be the following grammar:

$S \to CS$, $S \to AS$, $S \to CA$, $S \to DD$, $S \to AC$,

$C \to AA$, $C \to BB$, $D \to AA$, $D \to DC$,

$A \to a$, $B \to b$.

$N = \{S, C, D, A\}$, Ter $= \{a, b\}$. Let $w = aabba$. The parsing table is presented in Fig. 2 and the parsing tree in Fig. 1.



Node	Label	Father	Left	Right
1	S	-	2	5
2	C	1	3	4
3	A	2	-	-
4	A	2	-	-
5	S	1	6	9
6	C	5	7	8
7	B	6	-	-
8	B	6	-	-
9	A	5	-	-

Fig. 1. Parsing tree and its representation.

Let G be a directed acyclic graph given by the relation R, where $R(u, v)$ holds whenever (u, v) is an edge of G. We say that G satisfies the unique path condition (UPC, for short) iff, for every two nodes v and u, there is at most one path from u to v. (Equivalently, one can say that G is weakly acyclic, after removing the orientation the undirected graph is acyclic.) The best known upper bound for computing the transitive closure R^* of directed graphs is $\log^2 n$; however, if the graph satisfies the UPC, then this bound can be improved.

Lemma 1. *If the directed graph G with m nodes satisfies the UPC, then the transitive closure of G can be computed in* log m *parallel time on a P-RAM using m^3 processors.*

Proof. Let R be the relation corresponding to a directed acyclic graph G satisfying the UPC and let V be the set of nodes. Assume that the nodes are numbered $1, \ldots, m$. We say that a node is a sink iff it has outdegree zero. Let $s = \log m$. First we compute the tables $R_k[v]$ (corresponding to some relations) for $0 \leq k \leq s$.

$R_0 := R$;
for each sink v **do in parallel** $R_0[v, v] := \text{true}$;
for $k := 1$ **to** s **do**
  for each $v1$, $v2$, $v3$ such that $R_{k-1}[v1, v2]$ and $R_{k-1}[v2, v3]$
  do in parallel $R_k[v1, v3] := \text{true}$;

	0	1	2	3	4	5
0		A	C D		D S	S
1			A		S	S
2				B	C	S
3					B	
4						A
5						

Fig. 2. The parsing table for G and $w = aabba$.

We claim that there are no write conflicts in the above algorithm and that $R_s[v1, v2]$ holds iff there is a path from $v1$ to $v2$ and $v2$ is a sink.

The first fact follows from the following (easily provable) invariant: (*) *if* $R_k[v, v1]$ *and* $R_k[v, v2]$ *and* v, $v1$, $v2$ *are lying on the same path in* G, *then* $v1 = v2$, *for* $k = 0, \ldots, s$. This invariant implies that whenever we have $R_{k-1}[v1, v2]$ and $R_{k-1}[v2, v3]$ and $R_{k-1}[v1, v2']$ and $R_{k-1}[v2', v3]$, then $v2 = v2'$ because the UPC guarantees that all the nodes involved lie on the same path.

The second fact follows from our doubling technique. We are doubling the distances between $v1$, $v2$ for which $R_k[v1, v2]$ holds, until ultimately $v2$ becomes a sink. The following invariant can be easily proved: (**) *if* $R_k[x, y]$ *and* y *is not a sink, then* $\text{dist}(x, y) = 2^k$ (dist is the length of the path from x to y in the graph G).

We have computed a part of R^*; if y is a sink, then $R^*(x, y) = R_s[x, y]$. Now we compute R^* for all non-sink nodes. We introduce two relations R'_k and D_k ($k = 0, \ldots, s$), represented by two-dimensional tables with the same names. $R'_k[x, y]$ holds iff $R_k[x, y]$ holds and y is not a sink. $D_k[x, y]$ holds iff $\text{dist}(x, y) < 2^{k+1}$, for non-sink nodes x, y.

Let ID denote the identity relation and $\circ$ denote the composition of relations; we consider only the nodes which are not sinks. The relations D_k can be computed using the following recurrence formula (following from invariant (**)):

$$D_0 = R + \text{ID}, \qquad D_{k+1} = D_k + D_k \circ R'_{k+1}.$$

We can easily compute R'_0 and D_0, next we apply the recurrence equation $\log n$ times.

```
for k := 1 to s do
begin
  for each x, z do in parallel if D_{k-1}[x, z] then D_k[x, z] := true;
  for each x, y, z such that D_{k-1}[x, y] and R'_k[y, z] do in parallel
  D_k[x, z] := true
end.
```

There are no write conflicts here because if x, y, and z are lying on the same path and $R'_k[y, z]$ holds, then y is uniquely determined by x and z (as a node lying on the path from x to z, whose distance to z is 2^k). Observe that in this algorithm D_k could be replaced by D (in fact, the subscript k is not needed, though it helps to apply the recurrence formula directly).

Now we can compute $R^* = R_s + D_s$ in one parallel step. □

Let Q be a vector of the type array $[0, \ldots, n]$ of booleans. Assume that at least one entry of Q contains the value true. Define the operation $\text{first}(Q) = \min\{0 \leq k \leq n : Q[k] = \text{true}\}$.

Lemma 2. *The operation* $\text{first}(Q)$ *can be computed on a P-RAM in* $\log n$ *time using* $O(n)$ *processors if* Q *contains at least one entry containing the value true.*

Proof. We can assume that the length of Q is a power of two (otherwise we could add a suitable (at most linear) number of dummy elements containing the value

false). The processors are organized into a balanced binary tree T. The leaves of this tree correspond to elements $Q[0]$, $Q[1]$, $Q[2]$, ... Denote by $T(v)$ the subtree rooted at the node v. If v is a node of T, then let $f(v)$ be the index of the first leaf in $T(v)$ containing the value true; if there is no such leaf, then let $f(v) = -1$. If v is a leaf corresponding to the ith element of Q, then

$$f(v) = (\textbf{if } Q[i] = \text{true } \textbf{then } i \textbf{ else} -1).$$

If v is a non-leaf node and $v1$ is its left son and $v2$ is its right son, then

$$f(v) = (\textbf{if } f(v1) \geq 0 \textbf{ then } f(v1) \textbf{ else } f(v2)).$$

Using this formula we can compute $f(v)$ for all nodes v level by level in a bottom-up manner. After this computation first$(Q) = f(r)$, where r is the root of the tree. We are making log n parallel steps using $O(n)$ processors. (In fact, $n/\log n$ processors could suffice here.) Obviously, there are no write conflicts in our algorithm. □

Theorem. *Assume that context-free recognition can be done in parallel time $T(n)$ using $R(n)$ processors of a P-RAM, and $T(n) = \Omega(\log n)$. Then the representation of a parsing tree (if there is any) can be constructed in parallel time $O(T(n))$ by using $O(R(n)n^2 + n^3)$ processors. If the recognition procedure constructs the parsing table, then $O(R(n) + n^3)$ processors are enough.*

Proof. First we construct the parsing table Tab for a given input string $w = a[1]a[2]...a[n]$ and a given grammar G in Chomsky normal form. This can be done n parallel time $T(n)$ using $n^2R(n)$ processors. For each A, $i < j$, we simultaneously check whether $A \to^* a[i+1]...a[j]$.

If $S \to^* w$, then we start to compute a parsing tree, else we stop here because we know that in such a case there is no parsing tree.

For a nonterminal A and sets of nonterminals $X1$ and $X2$, define the operation find$(A, X1, X2) = (B, C)$, where (B, C) is the lexicographically first pair of nonterminals such that there is a production $A \to BC$, $B \in X1$, and $C \in X2$. If there is no such pair (B, C), then find$(A, X1, X2)$ has the special value "undefined". The operation 'find' can be computed sequentially in $O(1)$ time by using one processor since the size of the grammar is bounded by a constant.

We now construct the following acyclic directed graph G represented by the relation R (the relation 'to be a possible father'). The set of nodes is

$$V = \{(A, i, j) : i < j, A \in \text{Tab}[i, j]\}.$$

With each pair (i, j), $0 \leq i \leq j \leq n$, is associated a vector mark$[i, j]$ of the type array $[0, \ldots, n]$ of booleans. All these vectors contain only values false initially. The initialization is made in one parallel step, for each entry simultaneously. Next we

execute the following algorithm:

for each node (A, i, j), $i < j-1$, **do in parallel**
begin
  for each $i < k < j$ **do in parallel**
    if find(A, Tab[i, k], Tab[k, j]) $\neq$ "undefined"
    then mark[i, j][k] := true;
  k := first(mark[i, j]); (B, C) := find(A, Tab[i, k], Tab[k, j]);
  {there is a production $A \to BC$ and $B \in$ Tab[i, k], $C \in$ Tab[k, j]}
  $R[(B, i, k), (A, i, j)] := R[(C, k, j), (A, i, j)]$:= true;
  {(A, i, j) becomes a possible father of (B, i, k) and (C, k, j)}
end.

It follows from the definition of the parsing matrix that for each $(A, i, j) \in V$ there exist suitable (B, i, k) and (C, k, j). The grammar does not have to be unambiguous. We choose the first suitable pair of nodes (B, i, k), (C, k, j) applying the operations 'first' and 'find'. The above algorithm works in log n time (we are in parallel applying the operation 'first' and logarithmic time follows from Lemma 2). There are no write conflicts.

The graph corresponding to our example grammar and the string *aabba* is shown in Fig. 3. Observe that the tree from Fig. 1 corresponds to a subgraph of this graph.

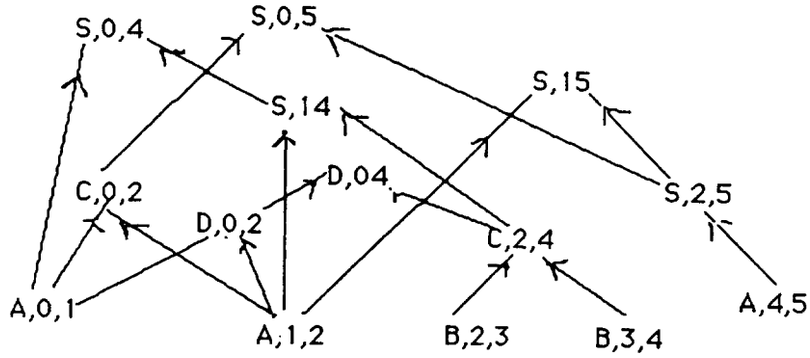


Fig. 3. The graph G.

Next we compute the transitive closure R^*. The graph G satisfies the UPC and we can use the algorithm from Lemma 1. Let $v_0 = (S, O, n)$. The parsing tree PT consists of all nodes v such that $R^*(v, v_0)$ holds. The root of PT is v_0. The function Father is computed as follows:

for each $u, v \in$ PT **do in parallel if** $R(u, v)$ **then** Father[u] := v;

The tables Left and Right can be computed in parallel time log n using the table Father. So far, the nodes are not numbered (from 1 to $2n-1$) as required, each node is a triple of the form (A, i, j) and all the tables have entries indexed by such triples. The set of such triples belonging to PT can be numbered from 1 to $2n-1$

using the algorithm of Tarjan and Vishkin [7] for preorder (or postorder) numbering of trees. This can also be done by arranging all possible triples in any initial order (e.g., lexicographical), and then the final number of a triple belonging to PT could be obtained by counting the number of preceding triples which are elements of PT (using a prefix computation). If 'num' is the numbering obtained, then Label[num(A, i, j)] := A. The constructed tree now satisfies all the requirements. □

It was proved in [5] that every unambiguous cfl can be recognized in log n time on a P-RAM using a polynomial number of processors. Now we can strengthen the result of [5].

Corollary. *Every unambiguous cfl can be parsed on a P-RAM in* log n *time using polynomial number of processors.*

Acknowledgment

The author thanks M.S. Paterson for his comments on this paper.

References

[1] W. Ruzzo, On the complexity of general context-free language parsing and recognition, in: H. Maurer, ed., *Proc. Internat. Conf. Automata, Languages and Programming*, Lecture Notes in Computer Science **71** (Springer, Berlin, 1979) 489–499.

[2] W. Ruzzo, Tree-size bounded alternation, *J. Comput. System Sci.* **21** (1980) 218–235.

[3] W. Rytter, The complexity of two-way pushdown automata and recursive programs, in: A. Apostolico and Z. Galil, eds., *Proc. NATO Advanced Research Workshop on Combinatorial algorithms on words* (Springer, Berlin, 1985) 341–356.

[4] W. Rytter, On the recognition of context free languages, in: A. Skowron, ed., *Computation Theory*, Lecture Notes in Computer Science **208** (Springer, Berlin, 1985) 318–325.

[5] W. Rytter, Parallel time log n recognition of unambiguous cfl's, in: L. Budach, ed., *Fundamentals of Computation Theory*, Lecture Notes in Computer Science **199** (Springer, Berlin, 1985) 380–389.

[6] W. Rytter and R. Giancarlo, Parallel parsing of bracket and recognition of input driven languages, manuscript, Univ. of Salerno, June 1985.

[7] R. Tarjan and U. Vishkin, Finding biconnected components and computing tree functions in logarithmic parallel time, *SIAM J. Comput.* **14**(2) (1985) 862–873.